Australian National University

The Department of Computer Science

Email Usage Stats Collector

Xiang Lv

Client/Supervisor: Eric McCreath

COMP8770 e-Research Project

June 2010

Table of contents

Acknowledgments 1

Abstract 2

1 Introduction 3

1.1 Overview 3

1.1.1 Background 3

1.1.2 Purpose 3

1.1.3 Statement of Scope 4

1.1.4 Project outputs 4

1.1.5 Target users 4

1.1.6 Participants 4

1.2 Introduction to Thunderbird extension 5

1.2.1 Overview of Thunderbird interface 5

1.2.2 Basic structure of Thunderbird extension 6

1.3 SQLite 7

2 **Requirements** 8

2.1 Operating system 8

2.2 Software Requirement 8

2.3 Languages 8

2.2 Summary of Requirements 8

3 **Schedule** 10

3.1 Initial plan 10

3.2 Actual progress 10

3.3 Ganett chart 11

4 **Model Design** 12

4.1 Context diagram 12

4.2 UML Diagram 13

4.3 Database Structure 16

5 **Implementation**.....19

5.1 Implementation phases 19

5.1.1 Modifying user interface 19

5.1.2 Defining application behavior 20

5.1.3 Operation of SQLite 21

5.1.4 Registering extension 23

5.1.5 Installing extension 24

5.2 User manual 25

5.3 Implementation Issue 25

5.4 Implementation Limitation 27

6 **Testing & Evaluation**.....28

6.1 Unit Test 28

6.2 Integration Test 29

6.3 Acceptance Test 32

7 **Conclusion & Future work**.....33

7.1 Conclusion 33

7.2 Future work 33

8 **Reference** 35

Appendix: User Manual..... 37

Acknowledgments

I would like to acknowledge my supervisor Dr. Eric McCreath who helped me over the entire period of this project. Whenever I had problems about the project, he would be ready to help me. Also he arranged the weekly meeting for me. He give me many good advice not only in the project implementation but also in writing the report and preparing the presentations.

I appreciate Uwe Zimmer who led the progress of the project. He arranged practice meeting almost every week, which helped me a lot in my project management.

I also appreciate the Mozilla Develop Centre, who provides many helpful documents, references, tutorials and sample codes about Thunderbird. I would not have finished the project without their help.

Abstract

Email is widely used in modern society. There is an increasing interest in understanding how different types of people use email. For example, how long does a user take to reply to messages, whether users would delete the messages before reading them, etc. The collector of these statistics would help researchers understand the behavior of users. The aim of this project is to implement an extension in Thunderbird that would enable the collection of such statistics.

This project uses Thunderbird, the email client software developed by Mozilla. Thunderbird is an open-source cross-platform software, so it may be easily extended. A Thunderbird extension, called “Stats_collector”, is developed to collect the statistics about email usage. The extension is built in the format which is associated with the XP Installer Package (.xpi) coming along with Mozilla products regardless of platforms. After the installation of the extension, the extension makes Thunderbird collect the statistics automatically.

This report describes the design and development of the “Stats_collector” for the Thunderbird system. Also the report details management and risks associated with the project. Unit and integration tests were performed to verify the quality of the implementation. The project is now at the stage where it could be incorporated into a user study.

1 Introduction

1.1 Overview

This section shows a basic overview of this project including the background information, purpose, scope and other related information.

1.1.1 Background

In 2008, more than 210 billion email messages were sent per day ([13]). And there are around 1.4 billion email users all over the world. So the large amount of email usage arises the interests of some researchers in understanding how different types of people use their email.

The statistics of the user events help researchers to indentify the behavior of email users. However, we are not aware of any software to collect such data. So there is an opportunity to develop an extension for the email client software which could collect the statistics automatically. And with the help of the extension, researchers would be able to obtain the data they are interested in.

In this project, Thunderbird, the email client software is adopted. It is an open source cross-platform email and Usenet client software developed by Mozilla. And it has become one of the most widely used email clients.

In this project, the objective is to build a Thunderbird extension called "Stats_collector" to store the email usage statistics and report the information to users or researchers.

1.1.2 Purpose

The main purposes of this project are listed below:

- **Collect usage event:** The system would accumulate Stats_collector when some user events happen, such as sending or receiving new messages. Then the detailed statistics would be automatically collected.
- **Store in SQLite database:** After collecting the statistics, Stats_collector should automatically store them into SQLite database in a well-formed format.

- **Query Function:** Stats_collector would provide users with some simple query functions. For example, it would show a brief summary of the messages, list the email user information and show the average time of replying to messages.

.

1.1.3 Statement of Scope

The statistics about the old messages which already exist in Thunderbird system are not expected to be collected by this extension. This is because the extension "Stats_collector" is accumulated by some user events, so it is impossible to deal with the old messages if no events happen with them.

A user study is not included in this project due to the limitation of time. The major task of the project is focused on the Thunderbird extension.

1.1.4 Project Artifact

The target outputs for this "Email usage statistics collector" is the Thunderbird extension "Stats_collector" in an xpi package with some user manual (given in Appendix) and the final project report.

1.1.5 Target users

The target users of this project are the Thunderbird users who are interested in identifying their own behavior of email usage. It would also help researchers to collect the email usage statistics of these Thunderbird users.

1.1.6 Participants

The participants of this project are described below:

Role	Name	Contact Details
Course coordinator	Uwe R.Zimmer	uwe.zimmer@ieee.org
Supervisor	Eric McCreath	ericm@cs.anu.edu.au

Developer	Lv Xiang	u4489326@anu.edu.au

1.2 Introduction to Thunderbird extension

A basic introduction to Thunderbird extension and how they are developed is now given.

1.2.1 Overview of Thunderbird Interface

Thunderbird contains many major components and correspondingly many XPCOM (Cross-Platform Component Object Model) interfaces.

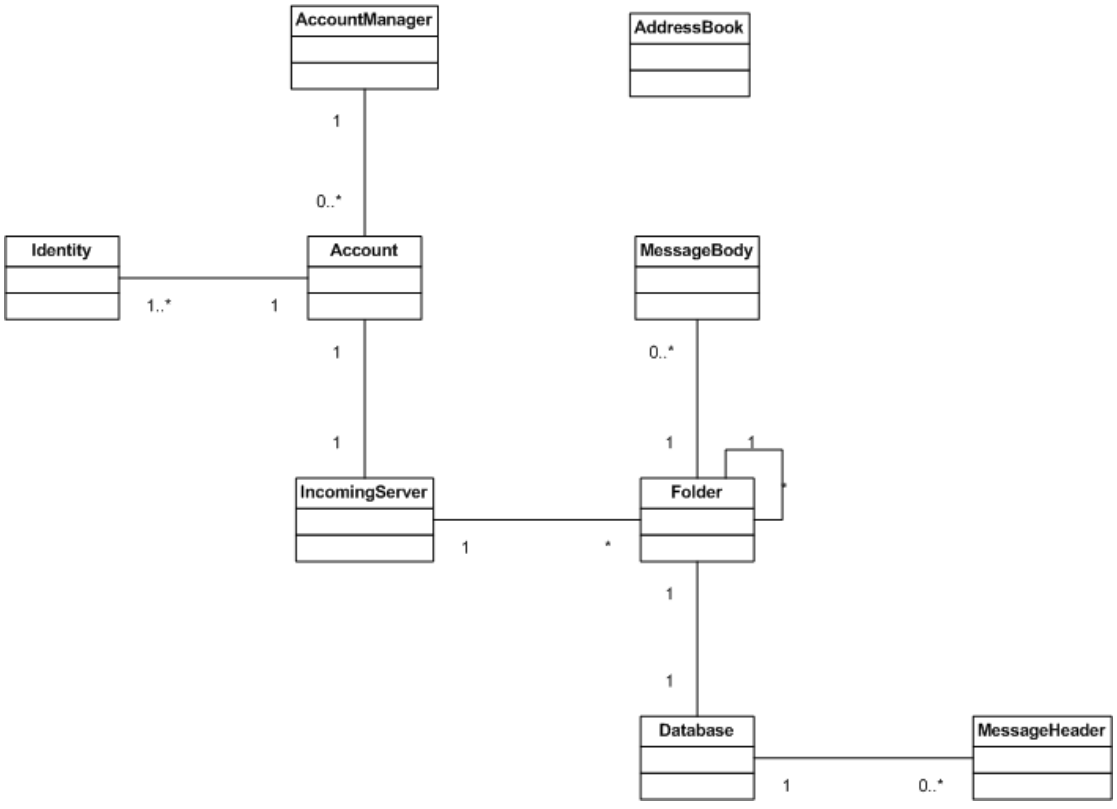

Figure 1.2 Basic overview of major thunderbird component (Source: Mozilla wiki)

Figure 1.2 shows the major components used in the Thunderbird system. Only the basic elements are listed in order to keep the whole diagram simple. Moreover, because the Thunderbird system is complex, so it indicates that only some of the interfaces would be implemented in this project. As shown in the figure, the interface MessageHeader and Folder are the main interfaces used for implementing in the

extension. The MessageHeader called “nsIMsgHdr” in the Thunderbird source code, which is a Thunderbird interface used to deal with the general information of the specific message such as the message common header and meta information about the message. The folder named “nsIMsgFolder” in the source code is also a Thunderbird interface to deal with the system folder such as Inbox or Outbox. The detailed information of such interfaces will be introduced later.

1.2.2 Basic structure of Thunderbird extension

The basic directory structure used for the Thunderbird extensions is listed below:

```
extension_file/
    chrome.manifest
    install.rdf
    content/
        overlay.js
        overlay.xul
    skin/
        overlay.css
    locale/
        overlay.dtd
```

This directory structure is detailed as follows:

1) **chrome.manifest** file demonstrates what packages and overlays are provided by the extension in the Thunderbird system.
2) **install.rdf** file contains the basic information of the extension. RDF is the abbreviation of “Resource Definition Framework”, Mozilla uses this format to store the register information and description data about the extension, such as the version, creator and id.
3) **Content** folder stores two kinds of files. The XML User Interface Language

(**XUL**) file is a XML-based language which would provide user interface widgets like buttons、menus and etc. In fact, the Thunderbird user interface itself is also written in XUL.

After adding the XUL elements, we could use **JavaScript** to define the actions of them. It is the core part of this project coding issue, because it defines the application's behavior.

4) Cascading Style Sheets (**CSS**) is a language which controls the display of XUL components.
5) Document Type Definition (**DTD**) is used to make the extension localizable. It would declare entities which are referenced in the XUL file.

1.3 SQLite

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. ([7]) Thunderbird provides a SQLite database API called "Storage" to implement the SQLite function in Thunderbird system. It contains several interfaces which could connect database, create and execute SQL statements on a SQLite database, etc.

2 Requirements

2.1 Operating system

Thunderbird is a widely used and cross-platform email client and works well on Windows, Linux and Macintosh. So it is a good base for us to develop an extension. The extension developed should work across these platforms.

2.2 Software Requirement

The main software and tools for this project are listed below:

- Thunderbird 3.0
- SQLite
- Venkman: a JavaScript debugger for Thunderbird
- Eclipse

2.3 Languages

Programming Language: JavaScript, XML

Database Query: SQLite

2.4 Summary of Requirements

The target user described detailed outline of email statistics collector project. And the summary of the requirements are listed below:

1. System Requirements: Stats_collector extension for Thunderbird should be easily to install.

2. Response to user event: Some user event such as sending or receiving email successfully should accumulate Stats_collector extension automatically.

3. Collect email statistics: Stats_collector should collect the related data such as sender, receiver, time, etc.

4. Store statistics in database: Stats_collector should create a SQLite database and store the statistics collected before. The database is supposed to contain six attributes: Sender, Receiver, Time, Subject, MessageID and Reference.

5. Show raw information about messages: Stats_collector should display all the messages received or sent by the Thunderbird user.
6. Calculate the average reply time: After user has input a specific contractor email address, Stats_collector should automatically display all the messages sent or received between it, and also generate the average reply time of the specific contractor. This function is accomplished through SQLite query.
7. Get brief summary information of specific contactor: This extension is supposed to get the total number sent or received with specific email address, and calculate the average message size.
8. Get brief information of message folder: Stats_collector should automatically store and represent the statistics about the Thunderbird message folders.

3 Schedule

3.1 Initial Plan

The initial planned timetable is shown below:

Stage	Week	Tasks
Plan	1-2	Project Selection and Requirements Specification
Background Research	3-5	Background research Learn how to build sample extension Prepare for initiate presentation
Design	6-7	Modeling and interface design
Implementation	8-11	Report outline Code implementation SQLite database implementation Develop Thunderbird extension “Stats_collector”
Test & Closure	12-13	Test Stats_collector Finish final report Prepare for final presentation

3.2 Actual progress

The actual timetable is shown below:

Stage	Week	Tasks
Plan	1-2	Project Selection and Requirements Specification
Background Research	3-4	Background research Learn how to build sample extension Prepare for initiate presentation
Design	5-6	Modeling and interface design
Understanding	7-8	Studying new technology Learn the source code and interface of Thunderbird Take references of other Thunderbird extensions
Implementation	9-11	Start Coding

		SQLite database implementation Develop Thunderbird extension “Stats_collector” Report outline
Test & Closure	12-13	Test Stats_collector Finish final report Prepare for final presentation

As the two tables shown in Section 3.1 and 3.2, the time used in background research saves around one week, because the test to build the very basic extension uses less time than anticipated. However, before implementation of code, some new technologies are required to be learned such as the C language. The source code of Thunderbird is written in C. To use the interfaces of Thunderbird system, extension developers must have a relatively clear understanding about the source code. In addition, the relationship between interfaces in Thunderbird is more complex than anticipated, so I spend extra time on identifying this relationship. And the new stage called “Understanding” covers all the tasks I have not planned before, which costs me two extra weeks. The newly added stage results in one week lost in the “Implementation” stage.

3.2 Gantt chart (Actual progress)

Figure 3.3.1 and Figure 3.3.2 shows the Gantt chat of actual progress and planned timetable correspondingly.

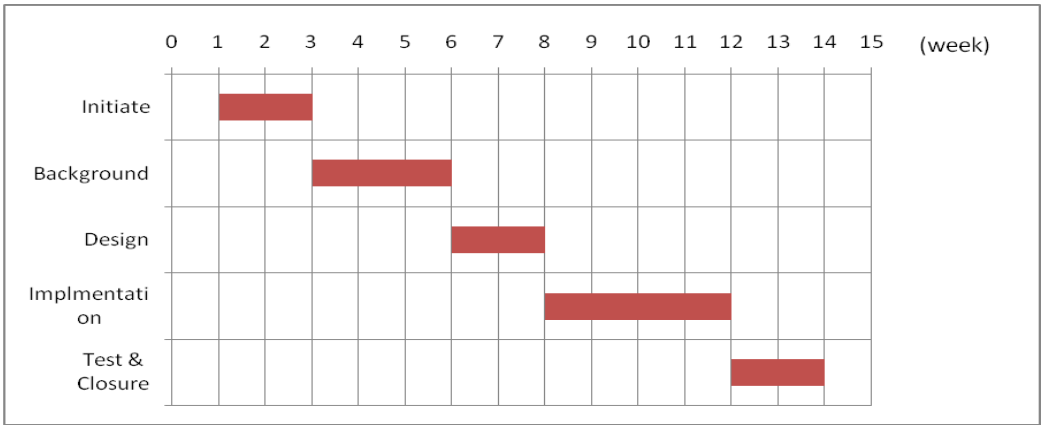

Figure 3.3.1 Planned Timetable Gantt Chart

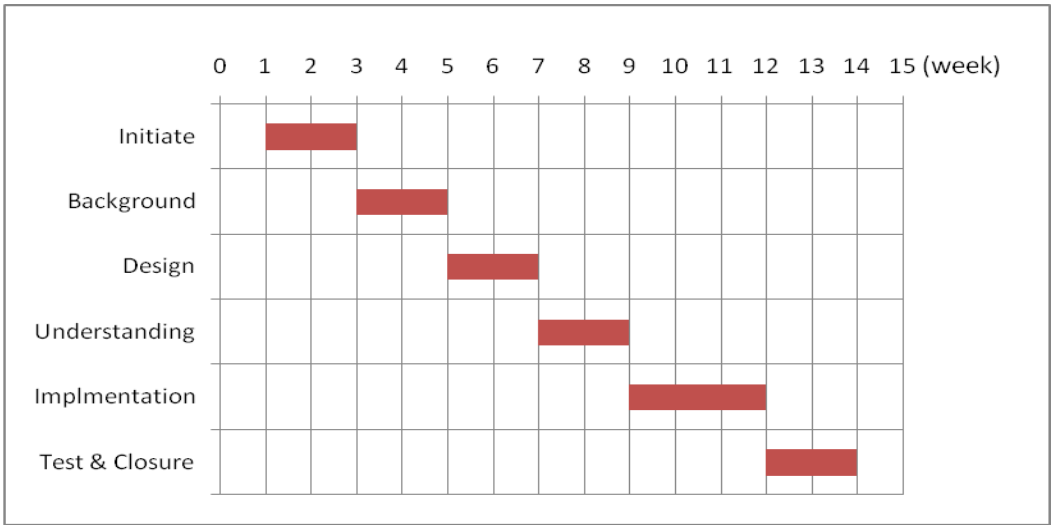

Figure 3.3.2 Actual Timetable Gantt Chart

4 Model Design

4.1 Context diagram

Figure 4.1 shows the context diagram of the main Thunderbird interface used in this "Email statistics collector" project.

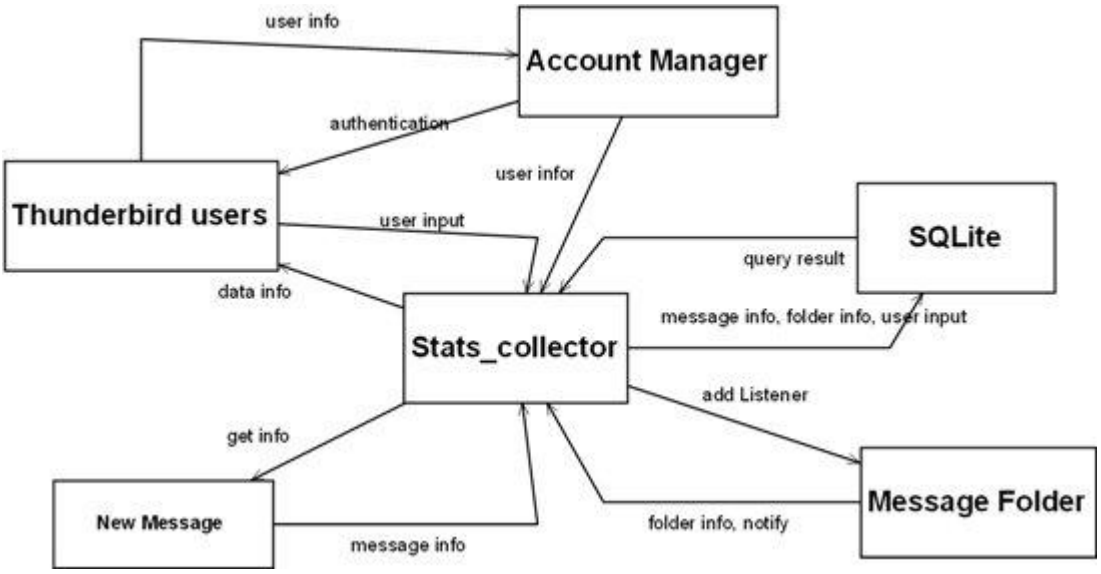

Figure 4.1 Context diagram of Stats_collector

So there are mainly five objects interacting with Stats_collector extension:

- **Thunderbird Users**

  Users should enter their email account and password when they login into the Thunderbird system. Then the Thunderbird account manager would collect the user information and pass it to Stats_collector later. Account manager would identify the user information and give authentication to the users.

  Users would choose some functions provided by the Stats_collector, and maybe some functions would produce promote dialog boxes to collect user information. Stats_collector would store the users input as parameters and provide users with the data information they require.

- **Account Manager**

  It stores the identified user information of the Thunderbird system. Users log in the Thunderbird through this interface and also the Stats_collector collects the user information through it.

- **New message**

  The new messages represent the newly incoming and out going messages. When users inquire about the statistics of the message, this extension would automatically collect the detailed information of these new messages, including sender, receiver, MessageID, time and reference (if it exists).

- **Message Folder**

  Stats_collector adds a listener to the message folder of the Thunderbird system. When the number of messages in the inbox folder or the outbox folder changes, then this indicates that the user has received or sent messages. The listener tells the Stats_collector to collect the information of the new messages. Moreover, the folder transports information to the Stats_collector, such as the number of unread message, average message size etc.

- **SQLite**

  Stats_collector uses a SQLite database. Each time the Stats_collector collects the information of new messages, it inserts these values into the database. If the data base does not exist, a new one is created automatically.

  When users request a summary, the Stats_collector will pass the information to the SQLite. And SQLite will execute some queries to accomplish such functions and take user input as parameters. Then it will return the query result to the Stats_collector.

4.2 UML Diagram

Figure 4.2 shows the class diagram of main Thunderbird interfaces which are used in the Stats_collector extension.

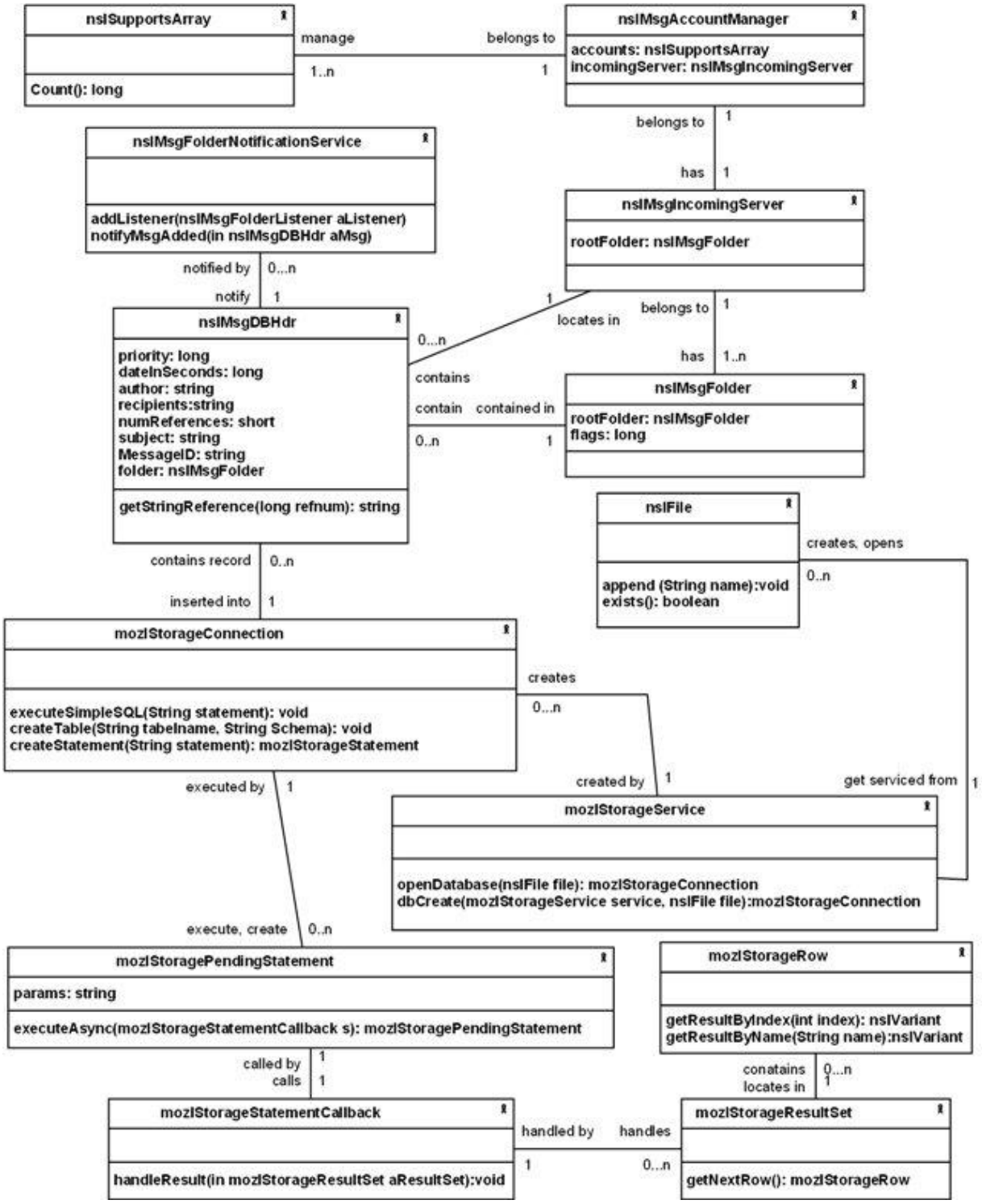

Figure 4.2 Class diagram of main interfaces used in Stats_collector

Thunderbird system has many interfaces; however, in this project only a few of them

are used. Some detailed information about the key interfaces used in Stats_collector extension are listed below:

- nsIMsgAccountManager

This interface deals with all the accounts which have been logged and created in local Thunderbird system. In the Stats_collector extension, this interface is used to get the account information and get access to the incoming server.

- nsIMsgFolder

Stats_collector collects the structure of the user account folder through the incoming server. This is focused on the inbox and outbox folder. Based on this interface, Stats_collector can collect information from each folder.

- nsIMsgDBHdr

This interface deals with the head of message. Stats_collector is activated to collect statistics when new messages sent or received. Then the Stats_collector will pass the information of new message to mozIStroageConnection as parameters, and mozIStroageConnection will insert the related value into SQLite database using the query statement.

- mozIStorageService

This interface is intended to be implemented by a service that can create storage connections to a database or a file. In this extension, Stats_collector is required to create a “thunder_info” table, so that it could insert values into the table. If the table does not exist, a new one is created. After the database is created, it also manages the access to open the database.

- mozIStorageConnection\mozIStoragePendingStatement\mozIStorageCallback\mozIStorageResult\mozIStorageRow (all related with the SQLite issues)

MozIStorageConnection represents a connection which attaches to a file or database.

The connection is created when mozIStorageService opens the database. In Stats_collector extension, is used to attach to the table “thunder_info”.
In the Thunderbird system, mozIStorageConnection can be used to execute the SQLite statements, such as inserting new values and executing queries. In this extension, for example, the information of new messages are inserted into the table through this interface.
MozIStorageConnection can also execute mozIStoragPendingeStatement statements which are very similar with a SQL query. It could contain some parameters within it. For example, in this extension Stats_collector sometimes takes the user input as parameters.
Moreover, MozIStorageConnection provides the way to handle the query result through the mozIStorageStatementCallBack interface. The mozIStorageStatementCallBack interface provides a more powerful way to deal with the mozIStorageResultSet. It could implement the mozIStorageStatementCallBack interface and call the handleResult method. Then Stats_collector uses it to control the output of the result set of queries. Developers can choose to show the result by row or by attributes name.

4.3 Database Structure

The SQLite database used in Stats_collector is built to store the information of message and execute some query functions. The detailed structure of it is listed in Figure 4.3 which shows the structure of SQLite database used in this extension.

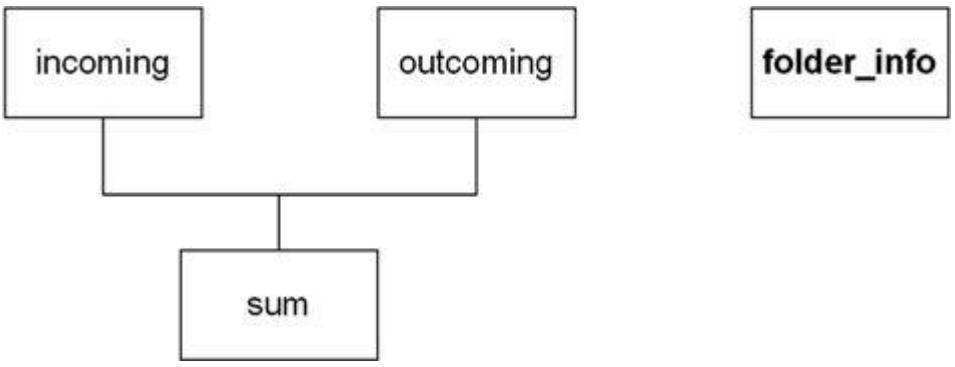

Figure 4.3 Basic structure of database

The details are illustrated in the table below.

Table Name	Comments
incoming	It stores the information of all the receiving messages.
outcoming	It stores the information of all the sending messages.
sum	It is a combination table of incoming and outcoming table.
folder_info	It stores the summary information of all the folder contained in the account, such as inbox and outbox, etc.

5 Implementation

5.1 Implementation Phases

The whole implementation phase could be divided into five phases. The details of these are listed below:

1) **Modifying user interface**: Implementing in XUL document to add the menu item in the menu bar.
2) **Defining application behavior**: Implementing JavaScript to define the function of XUL element.
3) **Operation of SQLite**: Create and maintain a SQLite database, and execute SQL statement base on it.
4) **Registering extension**: Implementing with install.rdf document to register extension in the Extension Manager run by Thunderbird.
5) **Installing extension**: Install the extension into Thunderbird.

5.1.1 Modifying user interface

In Thunderbird extension, XUL is used to modify the onscreen layout of the user interface and could be associated with JavaScript code to the interface. JavaScript could be used to define its functionalities. Developers could simply add the name of JavaScript function as a property.

For example, in my overlay.xul file:

```
<script src="overlay.js" type="application/javascript;version=1.7"/>
...
<menuitem id="folder" label="&folder_collect;"
          oncommand="folder_info_collect.onMenuItemCommand(event);"/>
```

The code simply adds a menu item and indicates that this element would correspond when it is clicked, and it would call the onMenuItemCommand function defined in the variable “folder_info_collect”.

After finishing editing the XUL file, I would define such overlays used in Stats_collector extension. Chrome.manifest is responsible for registering overlays.

For example:

```
overlay       chrome://messenger/content/messenger.xul
                                        chrome://helloworld/content/overlay.xul
```

It merges the overlay.xul I created with the default messenger.xul file. So "overlay.xul" file will be merged when messenger loads.

5.1.2 Defining application behavior

After finishing the implementation of user interface issue, I start coding the JavaScript file to define the behavior of each XUL element. Some functions are called when the user interface is loaded.

JavaScript file is the main part of this extension. It provides the functionalities of XUL elements and gets access to the SQLite database. In this project, the extension implements the interface of Thunderbird system, so that it can call the method and use the attributes. For example:

```
const Cc = Components.classes;
var dbService = Cc["@mozilla.org/storage/service;1"]. getService(Ci.mozIStorageService);
```

It indicates that the variable dbService implements the mozIStorageService, so that dbService could use the method defined in mozIStorageService, such as open a database.

```
var dbConnection = dbService.openDatabase(aDBFile);
```

After implementing the related interface, I start to collect the information. In this extension, the collect function is called when a new message sent or received. To indentify the condition, I added a listener to the interface at first.

```
function init()
{
    var notificationService =
   Components.classes["@mozilla.org/messenger/msgnotificationservice;1"]
   .getService(Components.interfaces.nsIMsgFolderNotificationService);
   notificationService.addListener(newMailListener, notificationService.msgAdded);

}
```

Then I could call the msgAdded function which use the new message header as a parameter, so that I could collect the information.

```
 msgAdded: function(aMsgHdr)
{
        var sender = aMsgHdr.author;
        var receiver = aMsgHdr.recipients;
        ...
        ...
}
```

5.1.3 Operation of SQLite

After collecting the statistics of the new message, I would use the database to store the related information, because it is relatively easy for developers to maintain and query. In Thunderbird system, SQLite database is applied. I would create a .sqlite file.

```
var dirService = Cc["@mozilla.org/file/directory_service;1"].  getService(Ci.nsIProperties);
var dbFile = dirService.get("ProfD", Ci.nsIFile);
dbFile.append("myproject.sqlite");
```

Then I could create the table:

```
dbConnection = this.dbCreate(dbService, dbFile); // get the storage connection
var createvalue = "CREATE TABLE IF NOT EXISTS tablename ('Sender' TEXT, 'Receiver'
TEXT,  'TIME' INTEGER,  'Subject' TEXT,'Reference' TEXT, 'MessageID' TEXT);";
dbConnection.executeSimpleSQL(createvalue);
```

In this extension, the table incoming, outcoming and sum have the same format. The basic structure of table is listed below:

Attribute name	Type	Comments
Sender	String	The author of message

Receiver	String	Recipients of message
TIME	Long	Time of message sent or received
Subject	String	The title of Message
MessageID	String	The unique String value of each Message
Reference	String	If the message is a reply message, it would contain the MessageID of the message it replies to.

Similarly, I use the same method to create the table folder_info to store the information of message folder in user account. The basic structure of this table is listed below:

Attribute name	Type	Comments
Name	String	The name of message folder, such as inbox or outbox.
Total	Long	The total number of messages contained in this folder.
Unread	Long	The number of unread message in the message folder.
Per	String	The percentage of unread message. Unread value divided by Total value.

Basically, SQL statements could be executed in two different ways in the JavaScript file. First one is to create a statement, and then just simply execute the statement. Second choice is to create mozIStoragePendingStatement, and use executeAsync() method, which I explain later. The first choice is relatively simple, but it could not introduce parameters, and it returns no value.

Now I can insert the information of message I collected before into the table.

```
var statement = dbConnection.createStatement
("INSERT INTO tablename VALUES(:sender, :receiver, :time, :subject, :reference, :id)");
statement.params.sender = sender;
statement.params.receiver = receiver;
statement.params.time = time;
statement.params.subject = subject;
statement.params.reference = reference;
statement.params.id = id;
statement.executeAsync();
```

The code listed below shows the second way to execute the SQL statements. It asynchronously executes the queries created with the connection. In this kind of way, I could bind parameters with the statement.

Finally, after inserting the data, I could perform some SQL queries on the dataset.

```
 var selectstatement = dbConnection.createStatement
("SELECT Sender, Receiver,TIME,Subject from try1");

selectstatement.executeAsync(
{
        handleResult: function(aResultSet)
        {

                ...
                ...
                for (let row = aResultSet.getNextRow();row;  row = aResultSet.getNextRow())
                {
                        var sender = row.getResultByName("Sender");
                        var receiver = row.getResultByName("Receiver");
                        var time = row.getResultByName("TIME");
                        var subject = row.getResultByName("Subject");
                        ....
                        ...
                }

        },

});
```

It still chooses the second way of executing select statements. Moreover it use mozIStorageCallback interface to handle the result set in order to control the view of final result presentation.

5.1.4 Registering extension

After finishing coding all the files listed below, I would register the extension. In Thunderbird extension, install.rdf is used by the Thunderbird Extension Manager when installing the extension. In this file, it mainly provides the metadata such as extension's version or author.

```
<em:id>u4489326@anu.edu.au</em:id>
<em:name>COMP8770</em:name>
<em:version>1.0</em:version>
<em:description>Stats_collector</em:description>
<em:creator>Lu Xiang</em:creator>
...
...
<Description>
      <em:id>{3550f703-e582-4d05-9a08-453d09bdfdc6}</em:id>
      <em:minVersion>3.0.0</em:minVersion>
      <em:maxVersion>3.0.*</em:maxVersion>
</Description>
```

In addition, I should also registers the extension content provider in the chrome.manifest file, so that the Extension Manager would find where the extension content locates in.

```
content      myproject    content/
```

5.1.5 Installing extension

There are basically two ways to install the extension.

The first way is to install this extension locally. First step is to locate extension folder of Thunderbird in the profile folder. For example, on a Mac platform will be ~/Library/Thunderbird/Profiles/extensions/… Second step is to create a text file in profile folder and put the full path of your extension inside, so that Thunderbird system could find the extension. Final step is to rename the text file with id of this extension. For example, in my extension, I use "u4489326@anu.edu.au" as the file name.

The second way is to package your extension in a "xpi" file and install it with the Extension Manager directly. One simple way to package into a xpi file is to use some software such as "WinRaR" and compress the extension into a "zip" file. Then you could simply rename the "x.zip" into "x.xpi".

The advantage of first way is to install locally and the extension is not managed by Thunderbird Extension Manager, so that I could modify the extension at any time. This way is relatively easy for debugging and testing. The second way is relatively easy for users to install and contribute, because only one "xpi" file is needed.

5.2 User manual

The detailed information about the functionality of the “Stats_collector” and how it is installed and used are given in the Appendix.

5.3 Implementation Issue

I have met two major issues while I do the implementation of the project.

Issue: It is relatively difficult to determine whether the message is a reply message and if it is, which message it replies to.

Resolution: It is difficult to determine whether the message is a reply message based on its subject. However, I could get the MessageID, In-Reply-To and References through nsIMsgDBHdr interface. Each message has its unique MessageID and each reply message got its “References”. The “References” is a list of MessageID which stores the “In-Reply-To” value when each reply action happens.

For example:

Message A:

Message-ID: <4BDFE476.9090602@anu.edu.au>

In-Reply-To: <4ee524.11e0c.1286270fd3a.Coremail.superbobo2@163.com>

References:

<4ee524.11e0c.1286270fd3a.Coremail.superbobo2@163.com>

Message B:

Message-ID: <f9616.12017.1286276021c.Coremail.superbobo2@163.com>

In-Reply-To: <4BDFE476.9090602@anu.edu.au>

References:

<4ee524.11e0c.1286270fd3a.Coremail.superbobo2@163.com>

<4BDFE476.9090602@anu.edu.au>

It indicates that the MessageID of the first original message is <4ee524.11e0c.1286270fd3a.Coremail.superbobo2@163.com>, and Message A is a

reply message to it. Message B also contains the MessageID of the original message, it indicates that it belongs to the same message collection with Message A. And its "In-Reply-To" value indicates that Message B is a reply message to Message A. Unfortunately, I could not find if there is a direct way to get the "In-Reply-To" value in nsIMsgDBHdr source code. But it could be identified that if the message is a reply message, the last value of its "References" attribute is always the same with its "In-Reply-To" value. So I could just use it to replace "In-Reply-To" value.

```
var Ref = aMsgHdr.getStringReference(aMsgHdr.numReferences - 1);
```

I could store the Ref as the "References" in the data table I created before. Based on it, I could identify the reply message when its References value matches the MessageID value of another message. Then I could calculate the average reply time of each user.

```
var selectstatement = dbConnection.createStatement
("SELECT AVG(incoming.TIME - outcoming.TIME) from incoming, outcoming
 WHERE incoming.Reference = outcoming.MessageID AND incoming.Sender = :sender");
selectstatement.executeAsync();
```

Issue: When the new message notifies the listener, it is a little difficult to identify whether the new message is sent or received. In the nsIMsgDBHdr interface, there is no direct method to identify the user event.

Resolution: I use the attribute "isRead" in nsIMsgDBHdr interface. Generally, when the incoming messages notify the listener, this attribute is always set as "false" value. However, the outcoming message is always set as "true" value. I use this method to give different flag the sent or received messages and insert the statistics into the corresponding table.

Issue: Some email accounts are not in a normative format, so it will be a little difficult when I store and query about the data. For example, the code could not identify that "John < u1234567@anu.edu.au>" is the same account with "u1234567@anu.edu.au".

Resolution: I remove the name of the contractor if it is there. So I added a few lines of codes in the JavaScript file to handle all the incoming or outcoming email address.

```
if ( str.indexOf("<") != 0)
{
        var ss = str.substring(str.indexOf("<")+1,str.indexOf(">"));
}
```

Before insert the values into the table, the extension should firstly identify whether in the email address contains the string "<", if contains just getting the substring between "<" and ">", then return the string values, and insert into the table.

5.4 Implementation Limitation

When I develop the extensions for Thunderbird, it is essential for me to work with the raw data in some non-user interface service. So this extension would have to get involved in XPCOM components provided by Thunderbird. And correspondingly XPConnect is also required to bridge between JavaScript and XPCOM components.

```
// use the components creted by Tunderbird
var dbService = Cc["@mozilla.org/storage/service;1"].  getService(Ci.mozIStorageService);

// create a new instance
Var MyXPCOM=Components.Classes[  "@mozilla.org/componentname]. createInstance();

//refer to interface
var account = accounts.QueryElementAt(i, Components.interfaces.nsIMsgAccount);
```

However, no integrated documents about the XPCOM components are available so that it is relatively difficult for me to use this component and its interface. I have to find some related extension of sample code to see how they implement. Moreover some Thunderbird XPCOM API documents which are available on the website are very limited, just listing the methods and attributes, but do not explain much. Another limitation is that no suitable developer tools are available. JavaScript in Thunderbird does not have a strict syntax. When error occurs, Thunderbird just ignores the error and continues the operation. Thunderbird does not provide developers with sufficient error message, for example, what is the type of error or where is the error. I have to insert many "alert" messages almost everywhere in the code, and try to identify where the error exists.

6 Testing & Evaluation

This section details the testing in this project. The tests of this project have three main sections: unit tests, integration tests and acceptance test. Unit tests are used to test each individual part of the "Stats_collector" extension, and integration tests are used to test the whole function of this extension. Acceptance test is provided by Dr.McCreath to check the whole quality of this project.

6.1 Unit Test

- **Test case1**

Name	Creation of table
Requirement	SQLite database "Stats_collect.sqlite" has been created and four table incoming, outcoming, sum and folder_info have been created.
Pre-conditions	Thunderbird is loaded.
Steps	1. Locate in the profile folder of Thunderbird 2. Check and open the SQLite file "Stats_collect.sqlite"
Result	"Stats_collect.sqlite" and its related table have been created accurately.

- **Test case2**

Name	Insertion of value
Requirement	New incoming and outcoming messages have been inserted into the table correctly.
Pre-conditions	Thunderbird is loaded and a new message is sent or received.
Steps	1. Use another email account to send the message into Thunderbird email account. 2. Check whether the information of this message has been inserted into table incoming and sum correctly. 3. Use the Thunderbird email account to send the message.

	4. Check whether the information of this message has been inserted into table outcoming and sum correctly.
Result	The statistics of the newly incoming and outcoming message have been inserted into the corresponding table accurately.

6.2 Integration Test

- **Test case3**

Case	Show raw data
Requirement	After clicking the button, Thunderbird will show a dialogue box contains all the information of message, such as Sender , Receiver, etc.
Pre-conditions	Thunderbird is loaded and at least one message has been sent or received.
Steps	1. Select in the Thunderbird menu bar. “Tools→ get user stats→ get statistics of all messages”. 2. Compare the contents shown in the dialogue box with the raw contents in your Thunderbird system.
Result	Thunderbird has shown the dialogue box and containing the exact contents. However, sometimes the contents are not shown in the right format, because some email address would have an extremely long length.

- **Test case4**

Case	Show specific result
Requirement	Thunderbird will show a dialogue box contains all the information of message related with a specific email address. And if the address is not in a right format, Thunderbird will alert an alarm message box.

Pre-conditions	Thunderbird is loaded.
Steps	1. Select in the Thunderbird menu bar. “Tools→ get user stats → specific email address”. 2. Enter the email address in the right format and click the confirm button. 3. Compare the contents shown in the dialogue box with the raw contents in your Thunderbird system. 4. Then enter the email address in the wrong format.
Result	When entering the right email address, Thunderbird will pop up a dialogue box containing the right information of the messages related with the email account. When enter the email address in the wrong format, it would pop up an alarm box. However, if the email address entered has never contact with the Thunderbird account, this extension will have no corresponding message.

- **Test case5**

Case	Show average reply time
Requirement	Thunderbird will show the average reply time of a specific email address. And if the address is not in a right format, Thunderbird will alert an alarm message box.
Pre-conditions	Thunderbird is loaded.
Steps	1. Select in the Thunderbird menu bar. “Tools→ get user stats → specific query → get average reply time”. 2. Enter the email address in the right format and click the confirm button. 3. Calculate the average reply time manually. 4. Compare the result with what you have calculated. 5. Then enter the email address in the wrong format.
Result	When entering the right email address, Thunderbird will pop up a

	dialogue box containing the right average reply time. And if inputting the wrong format, Thunderbird would pop up an alarm box.

- **Test case6**

Case	Show brief summary of specific contactor
Requirement	Thunderbird will show the brief summary information of a specific email address. And if the address is not in a right format, Thunderbird will alert an alarm message box.
Pre-conditions	Thunderbird is loaded.
Steps	1. Select in the Thunderbird menu bar. "Tools→ get user stats → specific query → count message amount". 2. Enter the email address in the right format and click the confirm button. 3. Calculate the total message number and average message size automatically. 4. Compare the result with what you have calculated. 5. Then enter the email address in the wrong format.
Result	When entering the right email address, Thunderbird will pop up a message box containing the right average message amount and message size. And if inputting the wrong format, Thunderbird would pop up an alarm box.

- **Test case7**

Case	Show brief information of message folder
Requirement	Thunderbird will show the brief statistic information of Thunderbird message folders in user account.

Pre-conditions	Thunderbird is loaded.
Steps	1. Select in the Thunderbird menu bar. “Tools→ get user stats → get message folder information”. 2. Compare the result with what you have counted yourself.
Result	Thunderbird will pop up a message box containing the right information of each folder in the Thunderbird system. However, due to limitation of JavaScript and Thunderbird storage dealing with the output representations, sometimes it will show in a slightly wrong format.

6.3 Acceptance Test

Acceptance testing for the extension “Stats_collector” will be checked by Dr. Eric McCreath. It would check whether the required functions have been implemented by this extension. This acceptance testing will be reflected in the artifact mark for the project.

7 Conclusion & Future work

7.1 Conclusion

The aim of this project is to develop a Thunderbird extension to collect the statistics of email usage and produce some simple summary message. The completion of this project required a basic understanding about XUL, JavaScript language, SQLite and the key interface of Thunderbird system. JavaScript is the main part of project implementation. However, the understanding of Thunderbird system is the most difficult part, because of the complexity of Thunderbird system and its incomplete reference documentation.

This report introduces the process to develop the Thunderbird extension, some key interface of Thunderbird system and the way of integrating SQLite database in Thunderbird.

The extension “Stats_collector” has the following functions:

- Store statistics of new message into SQLite database,
- Show the raw data of collected message,
- Show the messages contacted with specific email address,
- Calculate the average reply time of Thunderbird user and other contractors, and
- Show the summary information of account folders.

7.2 Future Work

“Stats_collector” is at an initiate stage to collect the statistics researchers need because of the limitation of time and Thunderbird extension resource.

The following tasks should be considered for future development of such extension:

- In this extension, Stats_collector could just get access to the new message, however, the information of old messages which have already existed could not be collected by Thunderbird. For new extension, it should involve the function of collecting the old messages.
- Stats_collector just correspond to the user event such as sending, receiving or deleting messages. However, it could not collect the statistics, for example, how

long does a user need to read a message?

- The user interface of extension is not well-designed so that users may feel confused when they use some functions. If possible, we could improve the whole extension to make it easier to understand and operate.
- The outputs of some functions are not in a good format. It just shows in a simple pop-up message box. If it possible, new extension would generate the output into a table format and users could choose to export the output into a new file.
- A user study is not included in this project, because the limitation of time. If it possible, this extension would be distributed in a reasonable way, and collect enough statistics for researchers to analyze the behavior of email users.

8 Reference

[1] Mozilla Developer Center, XPCOM Interface Reference, https://developer.mozilla.org/Special:Tags?tag=XPCOM+Interface+Reference, Accessed 20 April 2010

[2] Doug Turner, Ian Oeschger, Creating XPCOM Components, https://developer.mozilla.org/en/Creating_XPCOM_Components, Accessed 5 April 2010

[3] MozillaZine, Extension Development, http://kb.mozillazine.org/Extension_development, Accessed 2 March 2010

[4] JavaScript Tutorial, W3Schools Home, http://www.w3schools.com/js/default.asp, Accessed 14 April 2010

[5] Mozilla Developer Center, Storage, https://developer.mozilla.org/en/Storage, Accessed 2 May 2010

[6] Mozilla Developer Center, XPConnect, https://developer.mozilla.org/en/XPConnect, Accessed 14 April 2010

[7] SQLite home page, http://www.sqlite.org/, Accessed 5 May 2010

[8] Markus W.Scherer, BIDI algorithm for Mozilla, http://doxygen.db48x.net/mozilla-full/html/, Accessed 2 May 2010

[9] dianesun@netscape.com, sample code http://www.koders.com/javascript/fid04DE839424EB92BA8957421AE618056C1AB52CE4.aspx?s=search, Accessed 11 March 2010

[10]. Mozilla Developer Center, XUL Tutorial, https://developer.mozilla.org/de/XUL_Tutorial, Last accessed 22 April 2010

[11] Wikipedia, XPCOM, http://en.wikipedia.org/wiki/XPCOM, Accessed 1 April 2010

[12] Mozilla Developer Center, Compose New Message, https://developer.mozilla.org/en/Extensions/Thunderbird/HowTos/Common_Thunderbird_Use_Cases/Compose_New_Message#Open_New_Message_Window, Accessed 29 April 2010.

[13]. Mike Slocombe, 210 Billion Emails Sent Every Day, Digital lifestyle, 26th Jan 2009, From http://digital-lifestyles.info/2009/01/26/210-billion-emails-sent-every-day-70-is-spam, Accessed 20 May 2010.

[14]. By gfxl, Folders and Messages, Mozilla Developer Center, April 2010, https://developer.mozilla.org/en/Extensions/Thunderbird/HowTos/Common_Thunderbird_Use_Cases/Open_Folder#Watch_for_New_Mail, Accessed 21May 2010

Appendix: User Manual

A.1 Introduction to Stats_collector

Stats_collector is an extension of Thunderbird to collect the statistics of email usage. It would store the statistics into a SQLite database and execute some basic query function. To apply to this extension, the version of Thunderbird should be 3.0.0 or higher.

A.2 Extension installation

Step 1 : To store the extension, you should firstly start Thunderbird and click Tools→ Add-ons. (As shown in Figure A2.1)

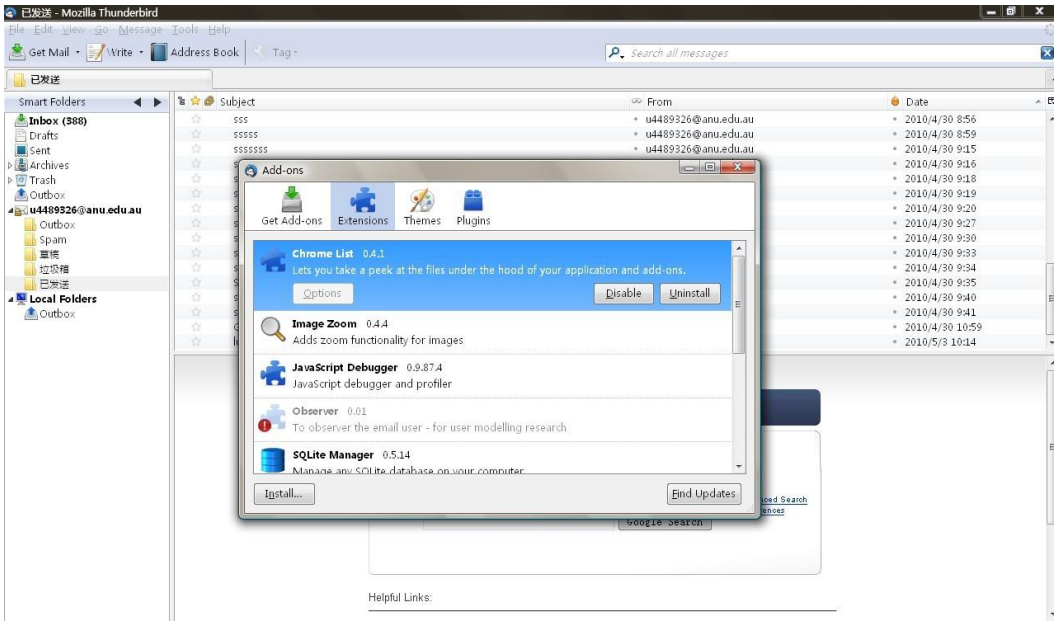

Figure A2.1

Step 2: click Install button and locate in the extension file, then confirm the installation. (As shown in Figure A2.2)

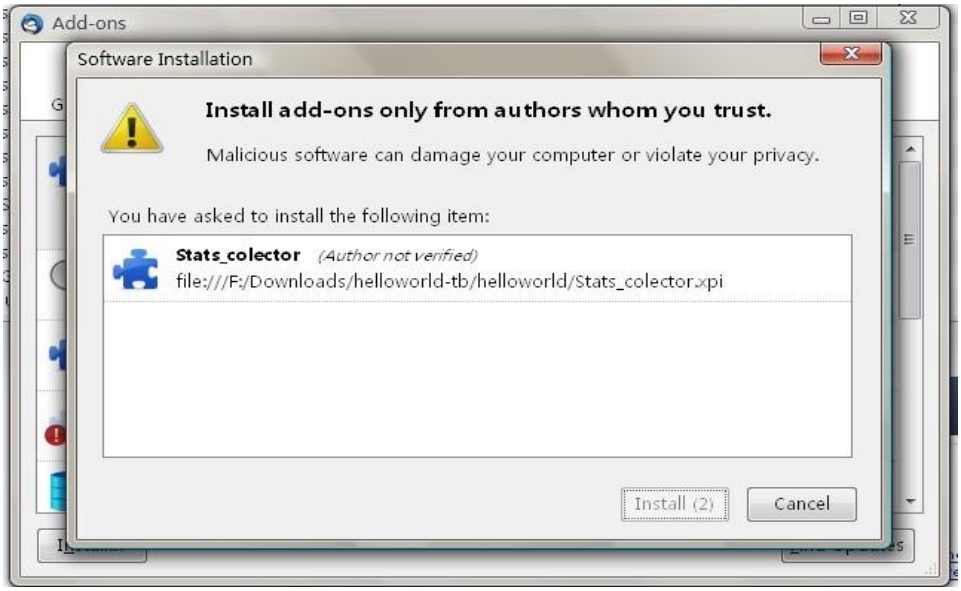
Add-ons
Software Installation
Install add-ons only from authors whom you trust.
Malicious software can damage your computer or violate your privacy.
You have asked to install the following item:
Stats_colector (Author not verified)
file:///F:/Downloads/helloworld-tb/helloworld/Stats_colector.xpi
Install (2)
Cancel

Figure A2.2

Step 3: After successfully installing the extension, Stats_collector would appear in the extension list. Figure A2.3 shows the information of all the installed extention.

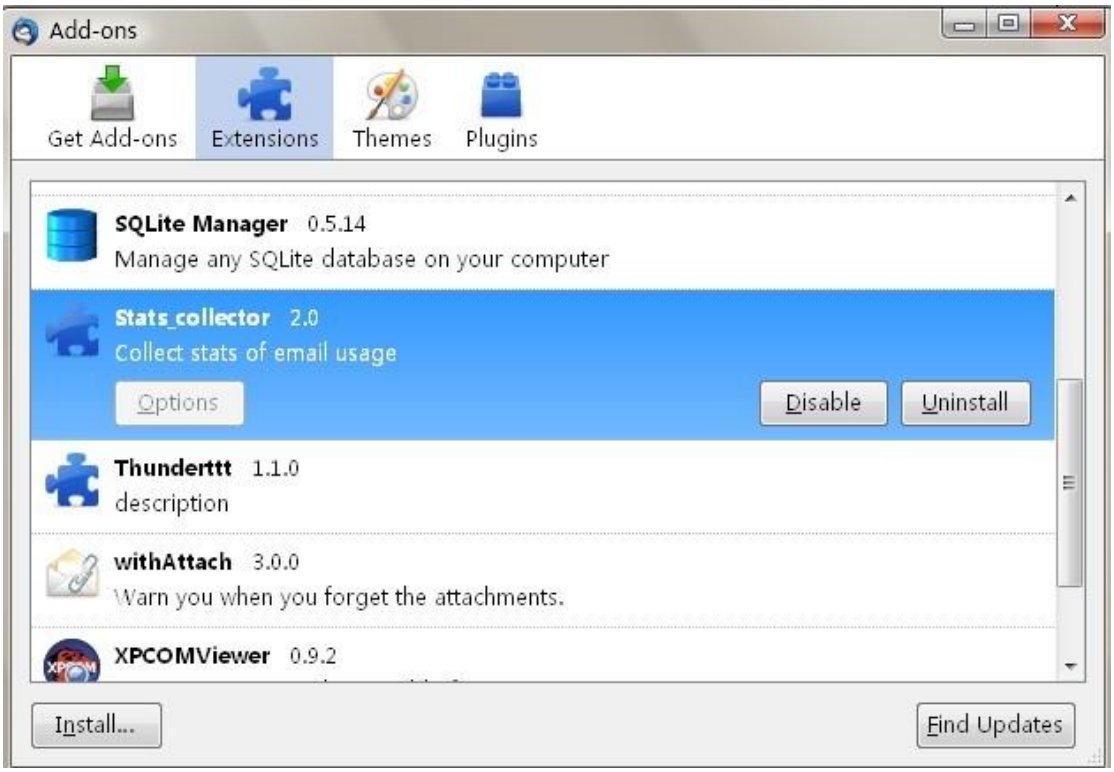

Figure A2.3 Thunderbird extension List

A.3 Walkthrough

- Get the brief statistics of all the new messages. Click “Tools-> Get user stats -> get stats of all message” Then Thunderbird will pop up a dialogue box which contains the information of all the new message collected by “Stats_collector”. (As shown in Figure A3.1)

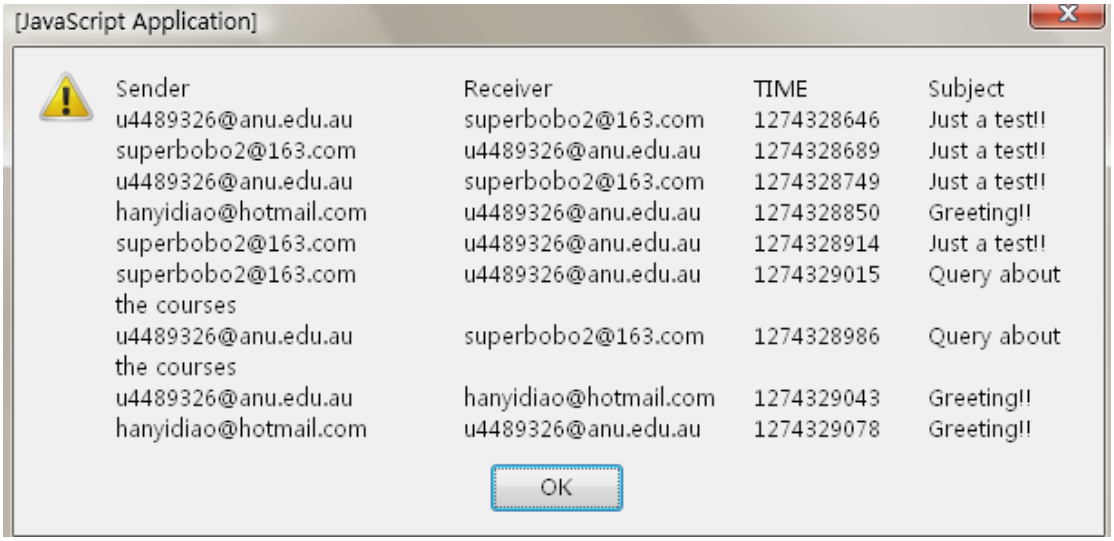
[JavaScript Application]
Sender Receiver TIME Subject
u4489326@anu.edu.au superbobo2@163.com 1274328646 Just a test!!
superbobo2@163.com u4489326@anu.edu.au 1274328689 Just a test!!
u4489326@anu.edu.au superbobo2@163.com 1274328749 Just a test!!
hanyidiao@hotmail.com u4489326@anu.edu.au 1274328850 Greeting!!
superbobo2@163.com u4489326@anu.edu.au 1274328914 Just a test!!
superbobo2@163.com u4489326@anu.edu.au 1274329015 Query about the courses
u4489326@anu.edu.au superbobo2@163.com 1274328986 Query about the courses
u4489326@anu.edu.au hanyidiao@hotmail.com 1274329043 Greeting!!
hanyidiao@hotmail.com u4489326@anu.edu.au 1274329078 Greeting!!
OK

Figure A3.1

- Get the information of the specific contactor. If users do not want to see all the messages, just want to see the message of specific contractor. Then click “Tools-> Get user stats -> specific email address”. Then enter the email address of the contractor you want to identify. Then Thunderbird will pop up a box contains the (As shown in Figure A3.2 & Figure A3.3) This extension would automatically check the format of email address, if users enter the wrong format, this extension would pop up an error message box.

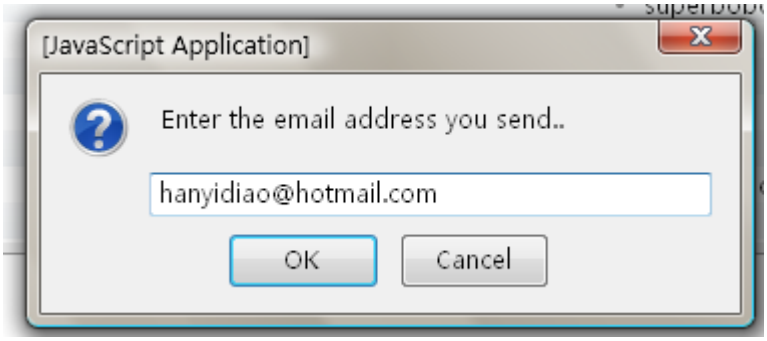

Figure A3.2

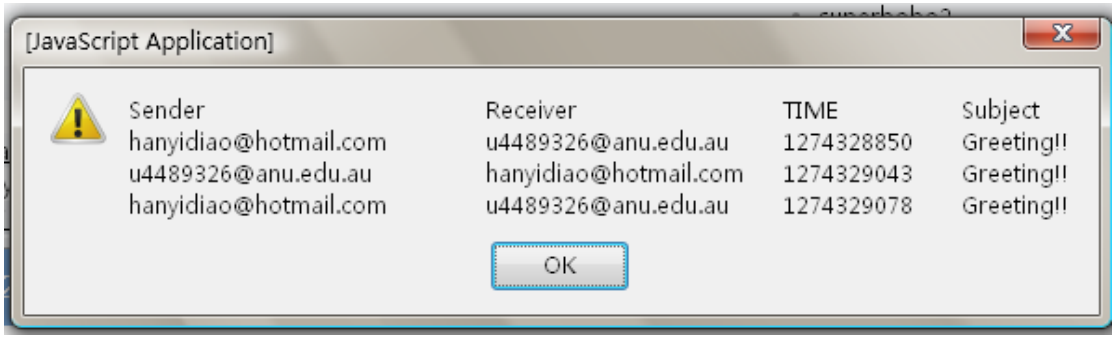

Figure A3.3

- Specific query. Get the brief summary information of a specific contractor. Users could click "Tools-> Get user stats -> `specific query`". It provides function of checking the average reply time of the specific contactor, the total number of messages contacted with it and the average message size. To use such functions, users could just enter the email address. Then Thunderbird would automatically pop up the corresponding message box with the information users query about. (As shown in Figure A3.4 , A3.5, A3.6 & A3.7)

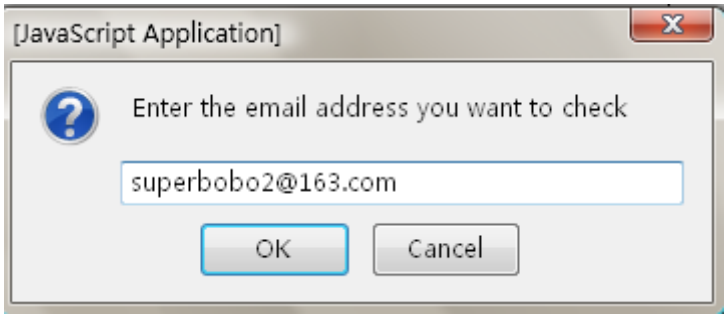

Figure A3.4

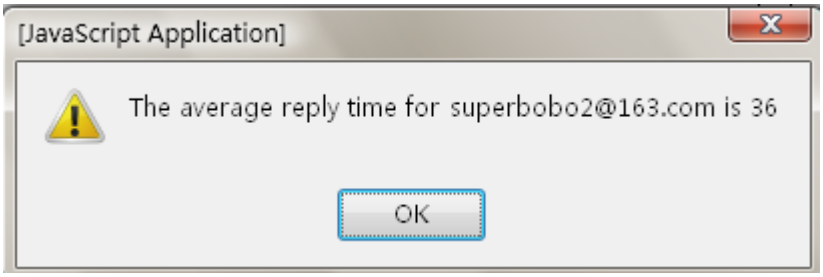

Figure A3.5

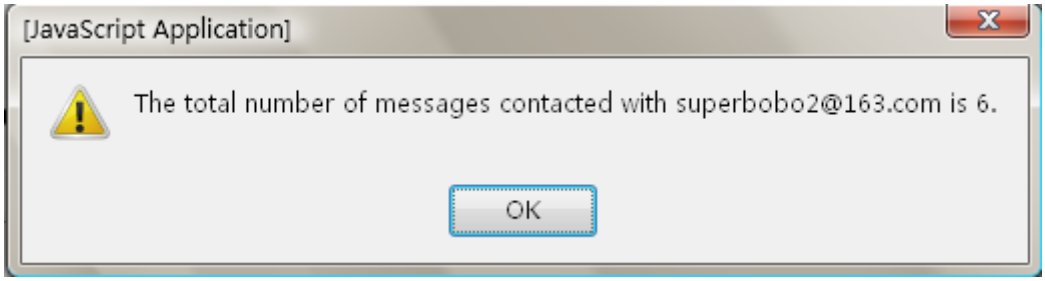

Figure A3.6

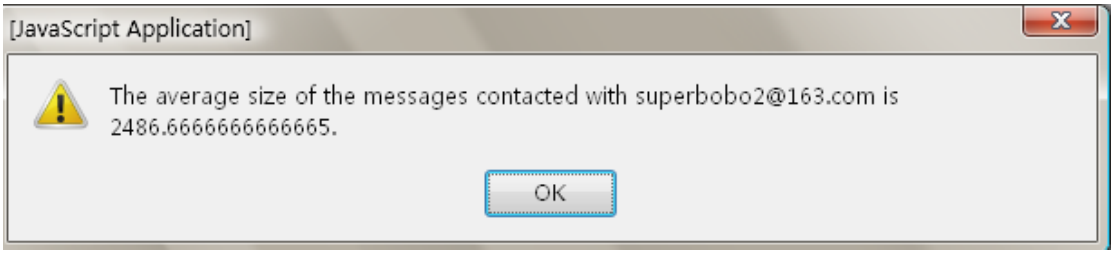

Figure A3.7

- Get the summary information of each folder in user account. Click "Tools-> Get user stats -> `get message folder information`". Then it will show the brief summary of each folder. (As shown in Figure A3.8)

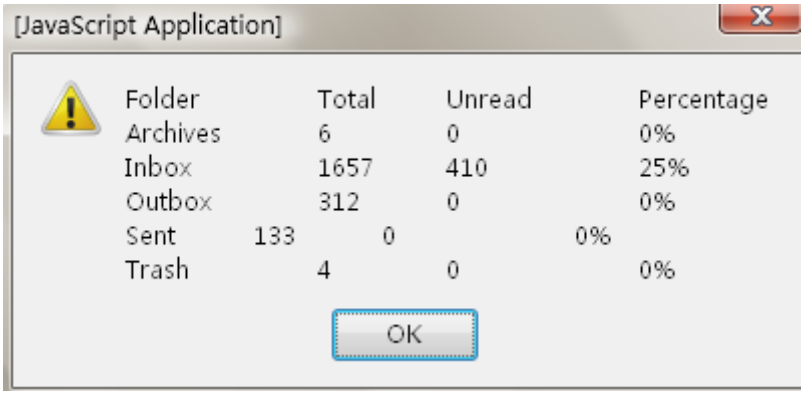

Figure A3.8